SYSTEM AND METHOD FOR CREATING INTELLIGENT SIMULATION OBJECTS USING GRAPHICAL PROCESS DESCRIPTIONS

CROSS-REFERENCE TO RELATED APPLICATION

This application is a continuation-in-part of application Ser. No. 12/284,662, filed Sep. 24, 2008, incorporated by reference in its entirety.

FIELD OF THE INVENTION

This invention relates to the field of computer modeling. More particularly, the invention relates to systems and methods for developing simulation.

BACKGROUND OF THE INVENTION

Over the 50 year history of discrete event simulation, the growth in applications has been facilitated by some key advances in modeling that have simplified the process of building, running, analyzing and viewing models. Three important advances have been: (i) the modeling paradigm shift from an event to a process orientation; (ii) the shift from programming to graphical modeling; and (iii) the emergence of 2D/3D animation for analyzing and viewing model execution. These key advances were made 25 years ago and provided the foundation for the set of modeling tools in wide use today.

The past 25 years has been a period of evolutionary improvements with few significant advances in the core approach to modeling. The currently available tools are mostly refined versions of what existed 25 years ago.

Many popular programming languages such as C++, C#, and Java are built around the basic principles of object oriented programming (OOP). In this programming, paradigm software is constructed as a collection of cooperating objects that are instantiated from classes. When instantiating an object into a model, one should start by specifying the properties governing the behavior of that object. For example, the properties for a machine might include its setup, processing, and teardown time, along with a bill of materials and the operator(s) required during setup. The creator of an object decides on the number and the meaning of its properties.

The typical instantiation of classes uses the core principles of abstraction, encapsulation, polymorphism, inheritance, and composition.

Abstraction can be summarized as focusing on the essential. The basic principle is to make the classes structure as simple as possible.

Encapsulation specifies that only the object can change its state. Encapsulation seals the implementation of the object class from the outside world.

Polymorphism provides a consistent method for messages to trigger object actions. Each object class decides how to respond to a specific message.

Inheritance allows new object classes to be derived from existing object classes, sometimes referred to as the "is-a" relationship. This is also referred to as sub-classing since a more specialized class of an object is being created. Subclassing typically allows the object behavior to be extended with new logic, and also modified by overriding some of the existing logic.

Composition allows new object classes to be built by combining existing object classes, sometimes referred to as the "has-a" relationship. Objects become building blocks for creating higher level objects.

Within this framework, objects are implemented by coding one or more methods that change the state of an object. Derived objects may override (i.e., replace) methods that are inherited from its parent class, or extend the behavior by adding additional methods.

The roots of these ideas date back to the early 1960's with the Simula 67 simulation modeling tool. That tool was created by Kristen Nygaard and Ole-Johan Dahl (1962) of the Norwegian Computing Center in Oslo to model the behavior of ships. Nygaard and Dahl introduced the basic concepts of creating classes of objects that own their data and behavior, and could be instantiated into other objects. This was the birth of modern object-oriented programming. Because Simula 67 was a programming language and not a graphical modeler, it never developed into a widely used tool.

In the early days of discrete event simulation, the dominant modeling paradigm was the event orientation implemented by tools such as Simscript (Markowitz et al., 1962) and GASP (Pritsker, 1967). In that paradigm, the "system" is viewed as a series of instantaneous events that change the state of the system. The modeler defines the events in the system and models the state changes that take place when those events occur. This approach to modeling, while very flexible and efficient, is also a relatively abstract representation of the system. As a result, many people found modeling with an event orientation to be difficult.

In the 1980's, the process orientation displaced the event orientation as the dominant approach to discrete event simulation. In the process view, one describes the movement of passive entities through the system as a process flow. The process flow is described by a series of process steps (e.g. seize, delay, release) that model the state changes taking place in the system. This approach dates back to the 1960's, with the introduction of GPSS (Gordon, 1960), and provides a more natural way to describe the system. Because of many practical issues with the original GPSS (e.g. an integer clock and slow execution), it did not become the dominant approach until improved versions of GPSS (Henriksen, 1976) along with newer process languages such as SLAM (Pegden/Pritsker, 1979) and SIMAN (Pegden, 1982) became widely used in the 1980's.

During the 1980's and 90's, graphical modeling and animation emerged as key features in simulation modeling tools. Graphical model building simplified the process of building process models while graphical animation dramatically improved the viewing and validation of simulation results. The introduction of Microsoft Windows made it possible to build improved graphical user interfaces and a number of new graphically based tools emerged (e.g. ProModel and Witness).

Another conceptual advance that occurred during this time was the introduction of hierarchical process modeling tools that supported the notion of domain specific, process libraries. The basic concept here is to allow users to create new process steps by combining existing process steps. The widely used Arena modeling system of Pegden/Davis (1992) is a good example of this capability.

Since the wide spread shift to a graphics-based process orientation, there have been refinements and improvements in the tools but no real advances in the underlying framework. The vast majority of discrete event models continue to be built using the same process orientation that has been widely used for the past 25 years.